МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное   
учреждение высшего образования

**"Южно-Уральский государственный университет**

**(национальный исследовательский университет)"**

**Высшая школа электроники и компьютерных наук**

**Кафедра системного программирования**

ОТЧЕТ

о выполнении практического задания № 4

по дисциплине

«Теория, методы и средства

параллельной обработки информации»

|  |  |
| --- | --- |
|  | Выполнил:  студент группы КЭ-303  Старостенок Д.В.  Проверил:  доцент кафедры СП  Маковецкая Т.Ю. |

Челябинск-2023

Задание:

Изучите конструкции для управления работой с данными shared и

private. Напишите программу, в которой создается k нитей, и каждая нить выводит на экран свой номер через переменную rank следующим образом:

rank = omp\_get\_thread\_num();

printf("I am %d thread.\n", rank);

Экспериментами определите, общей или частной должна быть переменная rank.

Листинг программы:

int number\_threads\_3;

printf("Write number of threads for lab 4: ");

scanf\_s("%d", &number\_threads\_3);

int rank = omp\_get\_thread\_num();

#pragma omp parallel num\_threads(number\_threads\_3)

{

printf("I am %d thread\n", rank);

}

Результат выполнения программы с типом shared для rank (Рис. 1).

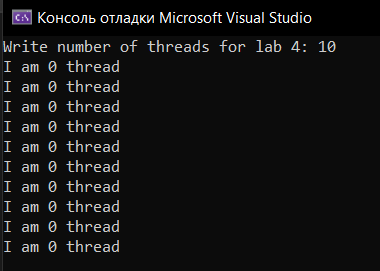


Рис. 1 – Результат выполнения программы shared

Результат выполнения программы с типом private для rank (Рис. 2).

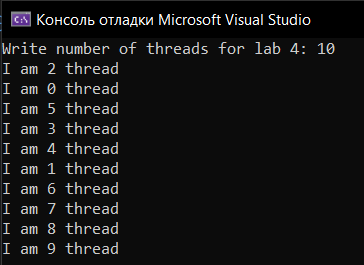


Рис. 2 – Результат выполнения программы private

Ответы на вопросы:

1) Для чего нужны частные переменные? Не противоречит ли их существование реализуемой OpenMP модели программирования в общей памяти? Приведите содержательный пример частной переменной.

Частные переменные необходимы для того, чтобы каждая нить имела свою локальную копию переменной внутри параллельного региона, а не общую. Это позволяет избежать некорректных результатов при одновременном обращении нескольких нитей к одной переменной.

Использование частных переменных не противоречит модели программирования в общей памяти, так как каждая нить имеет доступ только к своей локальной копии переменной внутри параллельной области.

Пример:

int k;

printf("Write number of threads for lab 4: ");

scanf\_s("%d", &k);

int rank;

printf("\npublic SLEEP:\n");

#pragma omp parallel num\_threads(k)

{

rank = omp\_get\_thread\_num();

Sleep(100);

printf("I am %d thread\n", rank);

}

printf("\nprivate:\n");

#pragma omp parallel num\_threads(k) private(rank)

{

rank = omp\_get\_thread\_num();

Sleep(100);

printf("I am %d thread\n", rank);

}

2) Какие новые области видимости появляются в параллельной программе? Как они задаются?

В параллельной программе появляются две области видимости: shared и private. Область видимости shared обозначает переменные, доступные для чтения и записи всеми нитями в параллельном регионе, а область видимости private обозначает переменные, которые должны иметь свою локальную копию для каждой нити в параллельном регионе.

3) Продемонстрируйте конфликт обращений к переменной rank в написанной программе? Всегда ли он возникает? Как его предотвратить?

Конфликт обращений к переменной rank возникает, когда несколько нитей пытаются одновременно обратиться к общей переменной rank без использования директивы private.

printf("\npublic SLEEP:\n");

#pragma omp parallel num\_threads(k)

{

rank = omp\_get\_thread\_num();

Sleep(100);

printf("I am %d thread\n", rank);

}

В таком случае значение переменной может быть изменено несколькими нитями одновременно, что приведет к некорректным результатам. Чтобы предотвратить конфликт обращений к переменной rank, необходимо использовать директиву private для того, чтобы каждая нить имела свою локальную копию переменной внутри параллельной области. Также можно использовать синхронизацию для контроля доступа к общей переменной.